Conditionals

Week 8

MEMS 1140—Introduction to Programming in Mechanical Engineering

Learning Objectives (L.O.)

At the end of this lecture, you should understand/be able to:

- ❏ What boolean values are;
- ❏ What conditional statements are;
- ❏ Write different types of conditional statements;
- ❏ Use conditional statements to control flow within a script.

Table of Contents (ToC)

1 – What are Boolean Values

A boolean value can occupy either **`true`** or **`false`**.

They are used to represent systems that can only be in one of two states.

For example, a (non-dimmable) lightbulb can only be on or off.

In code, this could be represented as follows:

```
light_bulb_on = true;
```

```
light_bulb_on = false;
```

2 – What are Conditional Statements

A conditional statement uses boolean values to control the actions within code.

The simplest statement flows as follows: **`if`** ... **`then`** ...

For example: **`if`** the light is on, **`then`** turn the light off.

In code, this would be written as follows:

```
if light_bulb_on
  turnBulbOff;
end
```

3 – The `else` Keyword

In addition to the **`if`** ... **`then`** ... flow, the keyword **`else`** can be used to control an action if the condition is ***not*** met.

For example:

```
if light_bulb_on
  turnBulbOff;
else
  turnBulbOn;
end
```

3 – `else if` Statements

To extend the logical flow beyond one check, **`else if`** can be used to add a followup condition to the first.

Perhaps you have a variable that can be checked in categories.

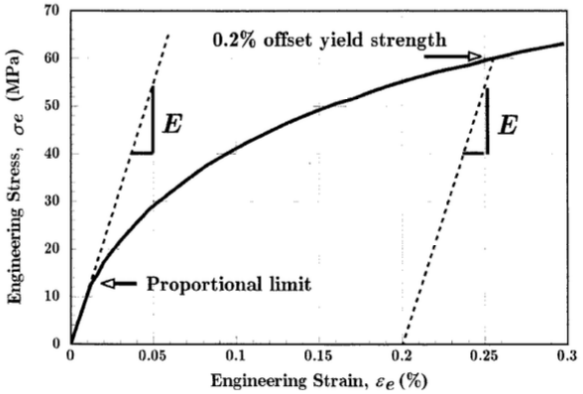
For example, it may be beneficial to consider regions in the stress-strain mechanical response for a material.

3 – Stress-Strain Regions

$$0 \leq \epsilon_e < \epsilon_{\text{elastic}}$$

$$\epsilon_{\text{elastic}} \leq \epsilon_e < \epsilon_{\text{yield}}$$

$$\epsilon_{\text{yield}} \leq \epsilon_e$$



3 – Stress-Strain Logic

Consider a strain gauge measurement.

Our code should identify which region of the graph the measurement occupies.

Note that the first two regions have lower and upper limits.

Logical operators `&&` (and) and `||` (or) are used to form more complex conditional statements, like comparing two bounds.

3 – And Statements — Both False

The **`&&`** operator results in an output of **`true`** when ***both*** of its inputs are **`true`**.

Input 1: **`false`** ***and*** Input 2: **`false`** → Output: **`false`**

```
>> a = 1;
>> b = 2;
>> c = (a < 0) && (b > 10)
c =
     logical
     0
```

Command Window

3 – And Statements — First True

The `&&` operator results in an output of `true` when ***both*** of its inputs are `true`.

Input 1: `true` ***and*** Input 2: `false` → Output: `false`

```
>> a = 1;
>> b = 2;
>> c = (a > 0) && (b > 10)
c =
     logical
     0
```

Command Window

3 – And Statements — Second True

The **`&&`** operator results in an output of **`true`** when ***both*** of its inputs are **`true`**.

Input 1: **`false`** ***and*** Input 2: **`true`** → Output: **`false`**

```
>> a = 1;
>> b = 2;
>> c = (a < 0) && (b < 10)
c =
     logical
     0
```

Command Window

3 – And Statements — Both True

The `&&` operator results in an output of `true` when ***both*** of its inputs are `true`.

Input 1: `true` ***and*** Input 2: `true` → Output: `true`

```
>> a = 1;
>> b = 2;
>> c = (a > 0) && (b < 10)
c =
     logical
     1
```

Command Window

3 – Or Statements — Both False

The `||` operator results in an output of **`true`** when ***at least one*** of its inputs is **`true`**.

Input 1: **`false`** ***or*** Input 2: **`false`** → Output: **`false`**

```
>> a = 1;
>> b = 2;
>> c = (a < 0) || (b > 10)
c =
     logical
     0
```

Command Window

3 – Or Statements — First True

The `||` operator results in an output of `true` when ***at least one*** of its inputs is `true`.

Input 1: `true` ***or*** Input 2: `false` → Output: `true`

```
>> a = 1;
>> b = 2;
>> c = (a > 0) || (b > 10)
c =
     logical
     1
```

Command Window

3 – Or Statements — Second True

The `||` operator results in an output of `true` when ***at least one*** of its inputs is `true`.

Input 1: `false` ***or*** Input 2: `true` → Output: `true`

```
>> a = 1;
>> b = 2;
>> c = (a < 0) || (b < 10)
c =
     logical
     1
```

Command Window

3 – Or Statements — Both True

The `||` operator results in an output of `true` when ***at least one*** of its inputs is `true`.

Input 1: `true` ***or*** Input 2: `true` → Output: `true`

```
>> a = 1;
>> b = 2;
>> c = (a > 0) || (b < 10)
c =
     logical
     1
```

Command Window

3 – Back to Stress-Strain

Return to the stress-strain example. This code determines which region of the curve the measurement occupies.

```
if (measured_strain >= 0) && (measured_strain < elastic_limit)
  % system undergoes elastic elongation
.
.
.
.
.
.
end
```

3 – Back to Stress-Strain

Return to the stress-strain example. This code determines which region of the curve the measurement occupies.

```
if (measured_strain >= 0) && (measured_strain < elastic_limit)
  % system undergoes elastic elongation
else if (measured_strain >= elastic_limit) && (measured_strain < yield)
  % system undergoes plastic elongation
.
.
.
.
end
```

3 – Back to Stress-Strain

Return to the stress-strain example. This code determines which region of the curve the measurement occupies.

```
if (measured_strain >= 0) && (measured_strain < elastic_limit)
  % system undergoes elastic elongation
else if (measured_strain >= elastic_limit) && (measured_strain < yield)
  % system undergoes plastic elongation
else if (measured_strain >= yield)
  % system is past its yield
.
.
end
```

3 – Back to Stress-Strain

Return to the stress-strain example. This code determines which region of the curve the measurement occupies.

```
if (measured_strain >= 0) && (measured_strain < elastic_limit)
  % system undergoes elastic elongation
else if (measured_strain >= elastic_limit) && (measured_strain < yield)
  % system undergoes plastic elongation
else if (measured_strain > yield)
  % system is past its yield strain
else
  % invalid measurement
end
```

3 – `switch` Statements

You can also use `switch` statements, which define cases for the test variable.

To replicate the light bulb example from earlier:

```
switch light_bulb_on
.
.
.
.
end
```

3 – `switch` Statements

You can also use **`switch`** statements, which define cases for the test variable.

To replicate the light bulb example from earlier:

```
switch light_bulb_on
  case true
    turnBulbOff;
  .
  .
end
```

3 – `switch` Statements

You can also use **`switch`** statements, which define cases for the test variable.

To replicate the light bulb example from earlier:

```
switch light_bulb_on
  case true
    turnBulbOff;
  case false
    turnBulbOn;
end
```

4 – Summary

This lecture covered:

- ✓ What boolean values are

  Booleans can be either **`true`** or **`false`**. They are used to reflect a case that can only occupy one of two states, like whether a lightbulb is on or off.

- ✓ What conditional statements are

  Conditional statements use the boolean results of one or more logical checks to control actions within a script. The most simple case follows the flow: **`if`** ... **`then`** ...

4 – Summary

✓ How to write different types of conditional statements

The **`else`** keyword can be used to increase functionality if the condition is *not* met. The **`else if`** keywords can be used to add a followup check to the first. And **`switch`** statements provide an alternative syntax for checking the status of a particular variable.

4 – Summary

✓ Using conditional statements to control flow within a script

By implementing conditionals, your script becomes more functional when dealing with different cases for certain variables. For example, considering whether your test specimen undergoes elastic or plastic strain, or even failure.